Complete Conversation: Capturing & Matching Spike Patterns

Dean: What I want to do is be able to capture spike patterns coming out of a neural network, out of a neuron when it fires. It may fire in some seemingly random pattern, but actually that pattern will repeat every time the conditions are the same for that neuron. So what I want to do is to be able to capture that spiking pattern, where the spikes are temporally separate and spaced out, and be able to capture that pattern so that in future experiences we want to be able to capture the similarity betwe...

Dean: Now I want to talk about what to do when, say, we set a maximum number of threshold patterns in our vector, reference patterns I mean, and we want to set a limit of, say, 10 reference patterns. Say we set the reference pattern vector at a maximum of 20 reference patterns. Now when another reference pattern comes in to be inserted, we have no more room. So I want to be able to come up with an algorithm that can blend in that new reference pattern into the closest similar reference pattern in the vector. How can we do that?

Assistant: Great question, Dean — you're basically asking how to incrementally merge a new pattern into the most similar existing reference pattern, once your max capacity (say, 20 patterns) is reached. This blends ideas from vector quantization, online clustering, and prototype-based learning (think: online k-means, LVQ, or SOMs). Let’s build the algorithm step by step.

Goal: Given: - std::vector> referencePatterns; (max size: 20) - A new incoming spike pattern std::vector newPattern

If referencePatterns.size() < 20: → Just add newPattern to referencePatterns.

Else: → Find the most similar existing pattern and blend the new one into it.

[Cosine similarity logic, blending function, and reference management strategy followed.]

Dean: Place all of these methods within the Neuron class and regenerate everything we've talked about to this point in that class.

Assistant: Here's a fully updated Neuron class that: 1. Maintains a rolling spike window based on timestamp. 2. Supports inserting new spikes while pruning old ones. 3. Learns new reference patterns. 4. Compares the current pattern against all learned reference patterns using cosine similarity. 5. Maintains a maximum number of reference patterns (default: 20). 6. Automatically blends new patterns into the closest existing one when the maximum is reached.

[C++ Neuron class implementation is included, as well as example usage.]